Още итеравивни алгоритми

Тодор Дуков

Подход при задачи с вече даден алгоритъм

Нека разгледаме следния алгоритъм:

```
int foo(int a)
{
    int x = 6, y = 1, z = 0;

    for (int i = 0; i < a; ++i)
    {
        z += y;
        y += x;
        x += 6;
    }

    return z;
}
```

Питаме се какво връща той?
Обикновено в такива задачи трябва да се изпробва алгоритъма върху няколко стойности. Можем да забележим, че:

- `foo(0)` връща `0`
- `foo(1)` връща `1`
- `foo(2)` връща `8`
- `foo(3)` връща `27`
- `foo(4)` връща `64`
- `foo(5)` връща `125`
- `foo(6)` връща `216`
- `foo(7)` връща `343`
- `foo(8)` връща `512`

Вече можем да видим какво прави алгоритъмът – `foo(a)` връща a^3. Нека сега докажем това:

Инварианта. *При всяко достигане на проверката за край на цикъла на ред* 5 *имаме, че:*

- $x = 6(1 + i)$
- $y = 3i^2 + 3i + 1$
- $z = i^3$

База. При първото достигане имаме, че:

- $i = 0$
- $x = 6 = 6 \cdot 1 = 6(1 + 0) = 6(1 + i)$
- $y = 1 = 3 \cdot 0^2 + 3 \cdot 0 + 1 = 3i^2 + 3i + 1$
- $z = 0 = 0^3 = i^3$

Поддръжка. Нека твърдението е изпълнено за някое непоследно достигане на проверката за край на цикъла. Тогава при влизане в тялото на цикъла:

- $\texttt{z}$ ще стане $\texttt{z} + \texttt{y} \overset{(\text{ИП})}{=} \texttt{i}^3 + 3\texttt{i}^2 + 3\texttt{i} + 1 = (\underbrace{\texttt{i} + 1}_{\text{ново i}})^3$
- $\texttt{y}$ ще стане $\texttt{y} + \texttt{x} \overset{(\text{ИП})}{=} 3\texttt{i}^2 + 3\texttt{i} + 1 + 6 + 6\texttt{i} = 3(\underbrace{\texttt{i} + 1}_{\text{ново i}})^2 + 3(\underbrace{\texttt{i} + 1}_{\text{ново i}}) + 1$
- $\texttt{x}$ ще стане $\texttt{x} + 6 \overset{(\text{ИП})}{=} 6(1 + \texttt{i}) + 6 = 6(1 + \underbrace{\texttt{i} + 1}_{\text{ново i}})$

Терминация. В последното достигане на проверката за край на цикъла имаме, че $\texttt{i} = \texttt{a}$, и тогава на ред 12 алгоритъмът ще върне $\texttt{z} = \texttt{a}^3$.
Нека разгледаме още един такъв пример:

```
1  int bar(int n)
2  {
3      return (int)sqrt((double)n * n);
4  }
5
6  int foo(int x, int y)
7  {
8      return (x + y + bar(x - y)) / 2;
9  }
10
11 int quux(int *arr, int n)
12 {
13     int a = arr[0];
14
15     for (int i = 1; i < n; ++i)
16     {
17         a = foo(a, arr[i]);
18     }
19
20     return a;
21 }
```

Искаме да видим какво връща $\texttt{quux}(\texttt{arr}, \texttt{n})$ където $\texttt{arr}$ е масив от цели числа а $\texttt{n}$ е броят на неговите елементи. Тук най-трудното, което трябва да се направи, е да се определи какво връщат $\texttt{bar}$ и $\texttt{foo}$:

- $\texttt{bar}(\texttt{n}) = \sqrt{\texttt{n}^2} = |\texttt{n}|$:
  - ако $\texttt{n} \geq 0$, то $\sqrt{\texttt{n}^2} = \texttt{n} = |\texttt{n}|$
  - ако $\texttt{n} < 0$, то $\sqrt{\texttt{n}^2} = -\texttt{n} = |\texttt{n}|$
- $\texttt{foo}(\texttt{x}, \texttt{y}) = \frac{\texttt{x}+\texttt{y}+|\texttt{x}-\texttt{y}|}{2} = \max\{\texttt{x}, \texttt{y}\}$:
  - ако $\texttt{x} \geq \texttt{y}$, то $\frac{\texttt{x}+\texttt{y}+|\texttt{x}-\texttt{y}|}{2} = \frac{\texttt{x}+\texttt{y}+\texttt{x}-\texttt{y}}{2} = \frac{2\texttt{x}}{2} = \texttt{x} = \max\{\texttt{x}, \texttt{y}\}$
  - ако $\texttt{x} < \texttt{y}$, то $\frac{\texttt{x}+\texttt{y}+|\texttt{x}-\texttt{y}|}{2} = \frac{\texttt{x}+\texttt{y}+\texttt{y}-\texttt{x}}{2} = \frac{2\texttt{y}}{2} = \texttt{y} = \max\{\texttt{x}, \texttt{y}\}$

Като знаем какво правят $\texttt{bar}$ и $\texttt{foo}$, можем да забележим, че $\texttt{quux}(\texttt{arr}, \texttt{n})$ дава най-големия елемент на $\texttt{arr}$:

Инварианта. *При всяко достигане на проверката за край на цикъла на ред* 15 *имаме, че* $\texttt{a} = \max \texttt{arr}[0 \ldots \texttt{i} - 1]$.

База. При първото достигане имаме, че $\texttt{a} = \texttt{arr}[0] = \max \texttt{arr}[0 \ldots 1 - 1] = \max \texttt{arr}[0 \ldots \texttt{i} - 1]$.
Поддръжка. Нека твърдението е изпълнено за някое непоследно достигане на проверката за край на цикъла. Тогава като влезем в тялото на цикъла, променливата $\texttt{a}$ става:

$$\begin{aligned}\texttt{foo}(\texttt{a}, \texttt{arr}[\texttt{i}]) \overset{(\text{ИП})}{=} \texttt{foo}(\max \texttt{arr}[0 \ldots \texttt{i} - 1], \texttt{arr}[\texttt{i}]) &= \max(\max \texttt{arr}[0 \ldots \texttt{i} - 1], \texttt{arr}[\texttt{i}]) \\ &= \max \texttt{arr}[0 \ldots \texttt{i}] = \max \texttt{arr}[\texttt{i} \ldots \underbrace{\texttt{i} + 1}_{\text{ново i}} - 1]\end{aligned}$$

Терминация. При последното достигане на проверката за край на цикъла на ред 15 променливата $\texttt{i}$ ще бъде $\texttt{n}$, откъдето функцията ще върне точно $\texttt{a} = \max \texttt{arr}[0 \ldots \texttt{n} - 1]$, с което сме готови.

Задачи

Задача 1. Даден е следният алгоритъм:

```
int num_slopes(int *arr, int n)
{
    int slopes = 1;

    for (int i = 1; i < n; ++i)
    {
        if (arr[i - 1] > arr[i])
            ++slopes;
    }

    return slopes;
}
```

Да се докаже, че при подаден масив от цели числа `arr` със размер `n`, функцията `num_slopes(arr,n)` връща броят на ненамаляващи подмасиви на `arr`.

Задача 2. Даден е следният алгоритъм:

```
int kadane(int *arr, int n)
{
    int m = arr[0], curr_m = arr[0];

    for (int i = 1; i < n; ++i)
    {
        if (arr[i] + curr_m > arr[i])
            curr_m += arr[i];
        else
            curr_m = arr[i];

        if (m < curr_m)
            m = curr_m;
    }

    return m;
}
```

Какво връща `kadane(arr,n)`, където `arr` е масив от цели числа с дължина `n`? Обосновете отговора си?

Задача 3. Даден е следният алгоритъм:

```
int find_majority(int *arr, int n)
{
    int maj = arr[0];
    int cnt = 1;

    for (int i = 1; i < n; ++i)
    {
        if (cnt == 0)
        {
            maj = arr[i];
            cnt = 1;
        }
        else
            cnt += (arr[i] == maj ? 1 : -1);
    }

    return maj;
}
```

Да се докаже, че при подаден масив от цели числа `arr` със размер `n`, в който има елемент с повече от $\lfloor \frac{n}{2} \rfloor$ срещания, функцията `find_majority(arr,n)` ще върне точно този елемент.